AD-A230 444

RADC-TR-90-282
Final Technical Report
December 1990

THE BBN KNOWLEDGE ACQUISITION PROJECT: PHASE TWO

BBN Systems and Technologies Corporation

DTIC
ELECTE
DEC 28 1990
D

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. 5290

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.
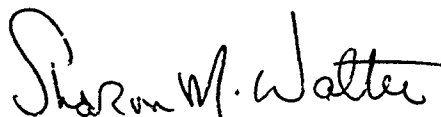
Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS) At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-282 has been reviewed and is approved for publication.
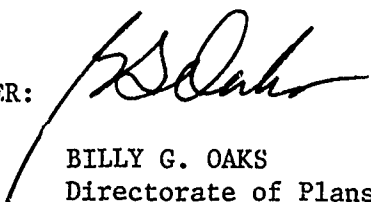
APPROVED: *Sharon M. Walter*

SHARON M. WALTER
Project Engineer

APPROVED: *Raymond P. Urtz, Jr.*

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: *Billy G. Oaks*

BILLY G. OAKS
Directorate of Plans & Programs

THE BBN KNOWLEDGE ACQUISITION PROJECT: PHASE TWO

Mark H. Burstein
Tom Reinhardt
Richard Shapiro

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE November 1990	3. REPORT TYPE AND DATES COVERED Final Jan 87 – Sep 89

4. TITLE AND SUBTITLE
THE BBN KNOWLEDGE ACQUISITION PROJECT: PHASE TWO

6. AUTHOR(S)
Mark H. Burstein, Tom Reinhardt, Richard Shapiro

5. FUNDING NUMBERS
C - F30602-85-C-0005
PE - 62301E
PR - E290
TA - 00
WU - 01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced   Rome Air Development Center (COES)
Research Projects Agency Griffiss AFB NY 13441-5700
1400 Wilson Boulevard
Arlington VA 22209-2308

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RADC-TR-90-282

11. SUPPLEMENTARY NOTES
RADC Project Engineer: Sharon M. Walter/COES/(315) 330-3577

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This document presents the final report on Phase Two of the BBN Knowledge Acquisition Project. It includes a brief overview of the project, and a review of KREME, the Knowledge Representation Editing and Modeling Environment developed during the project. KREME was designed to ease the problems involved in the development and maintenance of large knowledge based systems, to support experiments with knowledge acquisition and knowledge engineering techniques, and to provide a usable system for knowledge acquisition for knowledge based systems.

14. SUBJECT TERMS
Knowledge Base Editor, Knowledge Acquisition, Knowledge Representation, KREME, Frame

15. NUMBER OF PAGES
56

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

Table of Contents

List of Figures

Abstract

This document presents the final report on Phase Two of the BBN Knowledge Acquisition Project. It includes an overview of the project, and a review of KREME, the Knowledge Representation Editing and Modeling Environment developed during the project. KREME was designed to ease the problems involved in the development and maintenance of large knowledge based systems, to support experiments with knowledge acquisition and knowledge engineering techniques, and to provide a usable system for knowledge acquisition for knowledge based systems. During Phase One of the project KREME editors for Frames, Rules and Procedures were developed. Techniques were also developed for consistency maintenance and macro editing of knowledge bases. During Phase Two, refinements were made to the KREME Frame Language and Editor, with an eye toward real-world applications and portability. These refinements included extension of the frame language and the frame classifier, to allow for the expression of a wider variety of constraints on the represented knowledge. A new interface was also designed and implemented, enabling more flexible editing and browsing of knowledge bases. KREME was also converted to run in Common Lisp using the CLOS object system, in preparation for its use on other hardware platforms. Phase Two also included experiments in the use of KREME tools in the Common Lisp environment, in conjunction with a tool for document indexing and retrieval.

1

1. Introduction

This is the Final Report for Phase Two of the BBN Knowledge Acquistion Project. This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and was monitored by the Rome Air Development Center (RADC) under contract number F30602-85-C-0005.

The goal of this project was to create a useable and extensible knowledge engineering environment that would be capable of handling very large knowledge bases and support experiments with knowledge engineering techniques. During Phase One of the project we created KREME, the Knowledge Representation Editing and Modeling Environment. KREME is an extensible experimental environment for developing and editing large knowledge bases in a variety of representation styles. It provided tools for effective viewing and browsing in each kind of represenation, automatic consistency checking, macro-editing facilities to reduce the burdens of large scale knowledge base refision and some experimental automatic generalization and acquisition facilities. KREME was designed to facilitate the process of developing and editing representations of knowledge about a domain, while minimizing the classic problems of knowledge acquisition and knowledge base maintenance that arise during the development cycle of large expert systems. Knowledge engineers and subject matter experts with some knowledge of basic knowledge representation techniques will find it easy to use KREME to acquire, edit, and view from multiple perspectives knowledge bases that are several times larger than those found in most current systems.

During Phase Two of the project, our focus of attention was on refinements to the KREME Frame Editing environment, gearing that system to be even more useful in real-world tasks. We began by extending the semantic scope of the language so that knowledge engineers could more carefully encode constraints on the frames that they represented within a knowledge base for an application. We then rewrote the interface modules for the environment to take advantage of the capabilities and flexibility of a "desktop" style windowing environment, while at the same time introducing the notion of "window clusters" in order to retain the power of we had found in presenting multiple views on aspects of represented objects all at one time. We also converted the KREME system to CLOS and began experiments with integrating KREME's knowledge representation facilities into an environment requiring ongoing manual and semi-automatic knowlege acquisition and knowledge refinement, namely, a document indexing and retrieval system.

This report reviews our work on the revised version of **KREME**, BBN's **K**nowledge **R**epresentation, **E**diting and **M**odeling **E**nvironment, including our development of extended classification techniques and a new interface design for KREME. We also briefly describe our conversion of KREME to run under CLOS, the Common Lisp Object System, and an experiment in a more thorough integration of KREME into CLOS using the meta-object protocol. We conclude with a chapter on the some of the projects and systems that have adapted or incorporated KREME technology into them, to give a flavor for how it has been used.

The initial version of KREME was developed during Phase One by Mark Burstein, Glenn Abrett, Albert Boulanger and Richard Shapiro. That system included a family of editors in a single environment for knowledge

expressed in Frames, Rules and Procedures. The revised KREME frame editing system was written using CLOS, the Common Lisp Object System. A new interface, designed and developed by Richard Shapiro and Mark Burstein combined the flexibility of a "desktop" windowing environment with the capabilities of constraint frames composed of multiple windows. The conversion of the representation language and classifier from Symbolics Flavors to CLOS was done by the team of Richard Shapiro, Tom Reinhardt and Mark Burstein. The experiment with CLOS meta-object protocol was done by Tom Reinhardt.

Chapter 2 of this document reviews the functional structure of the KREME system, and describes the design of the revised KREME interface. Chapter 3 reviews our work on the extended classifier algorithm that supports a classification over a wider semantic base than the original KREME and NIKL languages did. Chapter 4 reviews our efforts at converting a portion of KREME to CLOS, and including an experiment in integrating KREME Frame semantics directly into the CLOS Meta-object protocol. For more information on these subjects, readers are referred to the Phase Two Technical Report [Burstein et al. 89a] and the KREME User's Manual [Burstein et al. 89b]. Chapters 5 gives a brief look at some of the uses that KREME technology has been put to. After a brief overview of KREME applications, we give a describe SFL and TARL, the representation tools for the SPROKET simulation environment. SFL and TARL are based directly on the KREME frame and procedures languages and editors. We also describe some KREME-related work on INDEXER, an on-line document retrieval system. Appendix A describes the Test Plan for the loading and startup of the revised KREME Frame Editor.

2. The New KREME Environment

In this chapter, we describe the overall design of the new KREME environment, and how it appears to the user. The first section gives an overview of the KREME architecture. The next section talks about the redesigned interface, and describes the appearance of the screen when using KREME. For a more complete description, see the KREME User's Manual [Burstein et al. 89b].

2.1 KREME Overview and Functional Structure

The current version of KREME provides, within a uniform environment, a number of special purpose editing facilities that permit knowledge to be represented and viewed in a variety of formalisms appropriate to its use, rather than forcing all knowledge to be represented in a single, unitary formalism. In addition to a general editing environment, KREME provides tools to do the kinds of validation and consistency checking so essential during the development or modification of knowledge bases. As the size of knowledge bases grows, and more people become involved in their development, this aspect of knowledge acquisition becomes increasingly important. In the hybrid or multi-formalism representational systems that are becoming prevalent [Rich 82, Brachman 83, Vilain 85], techniques must be provided for consistency checking not only within a single representational system, but between related systems.

Our approach to consistency maintenance has been to develop a *knowledge integration* subsystem that includes an *automatic frame classifier* and facilities for inter-language consistency maintenance. The frame classifier automatically maintains logical consistency among all of the frames or conceptual class definitions in a KREME frame base. In addition, it can discover implicit class relationships, since it will determine when one definition is logically subsumed by another even when the knowledge engineer has not explicitly stated that relationship. We also explored inter-language consistency maintenance facilities for detecting inconsistencies in references to frames in knowledge bases specified using other representation languages (e.g., rules, procedures). However, these tools are not yet ready to be included in the editing environment.

A second important area of investigation in developing the KREME editing environment has been the attempt to provide facilities for large-scale revisions of portions of a knowledge base. Our experience indicates that the development of an expert system inevitably requires systematic, large scale revisions of portions of the developed representation. This is often caused by the addition or redefinition of a task the system is to perform. These kinds of systematic changes to a knowledge base have, to date, only been possible by painstaking piecemeal revision of each affected element, one at a time. Our initial approach has been to provide a *macro-editing* facility, in which the required editing operations can be demonstrated by example and applied to specified sets of knowledge structures automatically. We plan to provide a library of such generic macro-editing operations for the most common and conceptually simple (though potentially difficult to describe) operations during phase two of the project.

Finally, we have begun to investigate techniques for *automatic generalization* of concepts defined in a knowledge base. We will briefly describe these experiments as well, in the section on *knowledge extension*.

A key design goal for KREME was to build an environment in which existing knowledge representation languages, appropriate to diverse types of knowledge, could be integrated and organized as components of a coherent global representation system. The current KREME Knowledge Editor can be thought of as an extensible set of globally coherent operations that apply across a number of related knowledge representation editors, each tailored to a specific type of knowledge. Our approach has been to integrate several existing representation languages in an open ended architecture that allows the extension of each of these languages. In addition, we have provided for the incorporation of additional representation languages to handle additional types of knowledge.

Underlying the entire system is a strong notion of meta-level knowledge about knowledge representation and knowledge acquisition. The representation languages were implemented based on a careful decomposition of existing knowledge representation techniques, implemented as combinable objects using FLAVORS [Keene and Moon 85] in the original version and CLOS [Bobrow et al. 88] classes in the new system. By organizing this meta-level knowledge base modularly, behavioral objects implementing such notions as inheritance and subsumption could be "mixed in" to a variety of representational subsystems making the incorporation of new representations and their editors a reasonably straightforward process. That is, each object in the meta-knowledge base encodes some aspect of a traditional representational technique, and is responsible for its own display, editing and internal forms.

The basic KREME frame-editing environment is composed of a number of components, designed to be reusable and reconfigurable for a variety of knowledge representation language editing tasks. The basic organization of the system as a set of modules is displayed in figure 2-1

Figure 2-1: KREME Functional Description

These pieces of functionality are organized as "mixins", reusable functional components that comprise the basic structures of frame, role, and slot definitions. Since KREME is built using an object-oriented substrate (Symbolics FLAVORs [Keene and Moon 85] in the original version, CLOS classes [Bobrow et al. 88] in the new version), these pieces of functionality are parts of the definitions of objects in the underlying object system. The basic classes of functionality are:

- Language Definition and Classification Objects
- Editor Interaction Objects (for Display and Editing Actions)
- Window Management Tools

The decomposition of the functionality of the basic representation, classification and editing facilities into their component object types is displayed in figure 2-2.

Figure 2-2: KREME Internal Object Functional Decomposition

As was the case with the original KREME system, these components can be reused with a variety of language definitions, so the basic editor facilities can (and have) been used for a number of different kinds of editing tools, including:

- A CLOS Browser
- An application-specific Rule Editor
- An editor for declarative representations of Frames, Goals and Procedures for a simulation environment
- Other application-specific editing tools.

For a more complete description of the original KREME design, see [Abrett and Burstein 87].

2.2 The New KREME Desktop Interface

The original KREME system was designed as a set of "views", each composed of a set of windows in a fixed configuration, occupying the whole workstation screen. Each view focused on one kind of representation, a concept, a role, a rule or a procedure, and filled the windows of the view with information about different aspects of that kind of representational object. Only one kind of knowledge representation object was presented at at time. Figure 2.2 shows an example of the old interface while editing a concept, as KREME Frames are called. Although the individual windows on each screen were general-purpose tools, it was impossible to do any reshaping of them to accomodate the varying amounts of information about any given representation object that was visible at one time. It was also impossible to bring up windows showing different objects or different kinds of objects at the same time.

When we converted KREME to run under CLOS, the Common Lisp Object System, we also redesigned the interface tools, decoupling them from the underlying representational systems, and redesigning the window tools to be more like those on an Apple Macintosh or a SUN workstation. The result was a more modular, and more flexible system for grouping movable, reshapable windows that allowed users individual preferences for editing become predominant.

As has become prevalent in many window-based interfaces, The KREME screen interface now appears to you as a **Desktop** covered by rectangular boxes called "windows" that can be moved around and stacked up like pieces of paper. Each window is independently movable and shapeable. Unlike most other window-based systems, KREME defines some sets of windows to act together, serving collectively to display and allow editing of a particular concept or role object. Sets of windows acting as a unit are called *clusters*. There are default arrangements for clusters for Concepts and Roles, and it is possible for the user to redefined the configuration of these window clusters in a file that is loaded before an editing session. We have tried to select and arrange the windows in each cluster definition so that you can edit a particular kind of knowledge representation object effectively and conveniently.

2.2.1 Component Windows

Figure 2.2.1 shows the as it might appear while editing the concept CAR. There are a number of different kinds of windows that can appear on the desktop. The ones shown are much the same as those shown in the prior figure. The difference is that these are only a few of the windows available, and they are all independently moveable. We first discuss what each of these windows are, and what they are for. The numbers in circles on this figure are referred to below, in describing that window.

The **Main commands window** (1) contains commands that may be invoked at any time while running KREME.

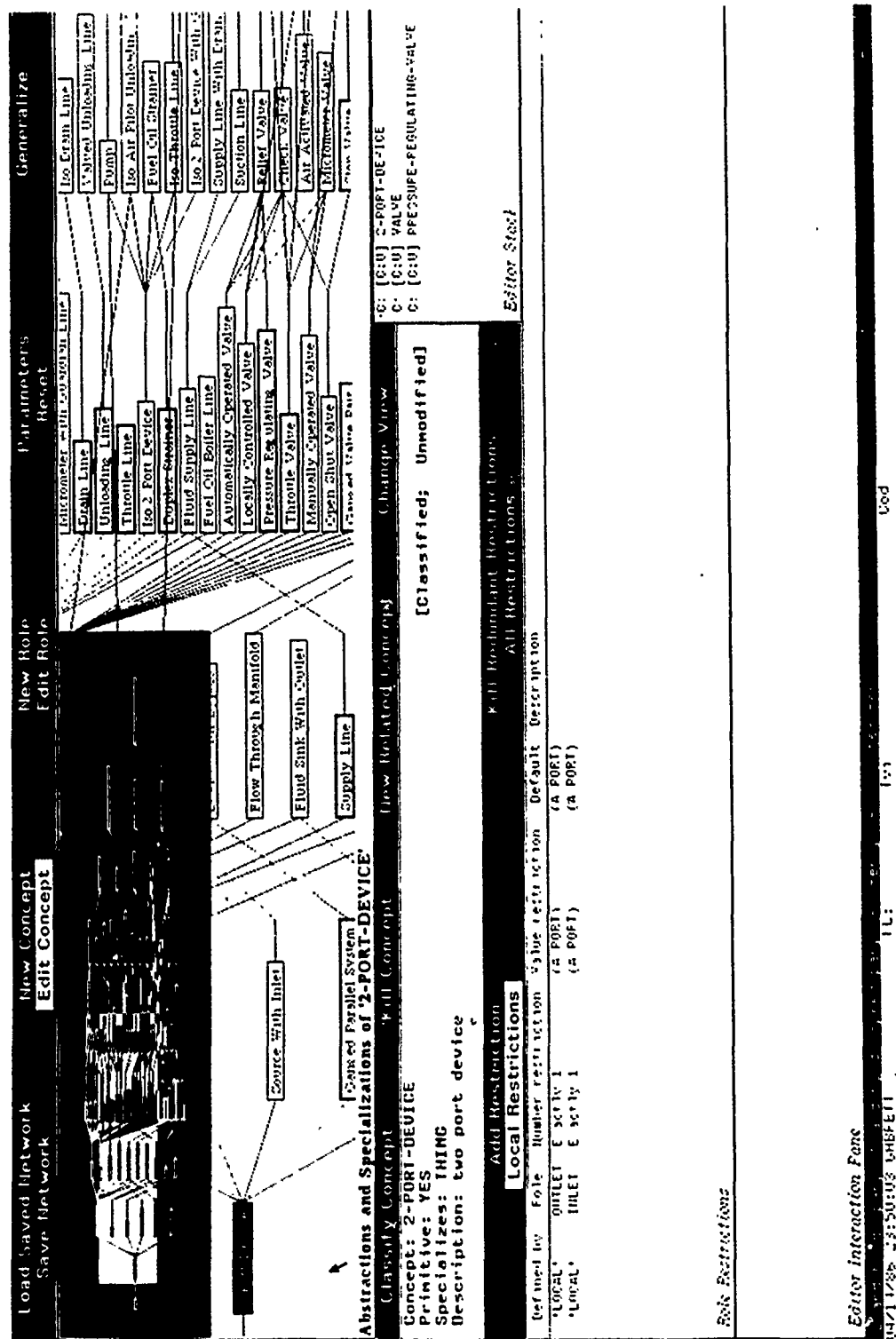**Figure 2-3:** Original KREME Editor when Editing a Concept

Figure 2-4: KREME's new Desktop Interface when Editing a Concept

The **Editor Items window** (2) shows the names of the things being edited states whether they are roles or concepts, and gives some information about their current edit state (e.g., whether they have been classified before, and modified since they were classified). This window is not normally visible, unless you click the <Middle Mouse> button on the desktop background.

The **Attributes window** (3) always displays pertinent information about the top item on the editor stack, which is called the *current editor object*. For concepts, the Attributes window contains the concept's name(s), a line specifying whether the concept is *primitive* and whether the concept has been *classified* (defined) or not, and whether it has been *modified* in the editor since it was last classified. It also includes lines giving the concept's parents, and a textual description. Variants of this window appear whenever you are editing a concept, a role, or a rule packet.

The **Graph window** (4) displays a dynamically updated graph of all of the abstractions and specializations of the current editor object. This view provides a constant visual display of the relative position of the object being edited in a hierarchy. Graph windows often appear when you are editing concepts or roles, or, in general, objects that live in hierarchies. There are also graphs for restrictions on slots, and graphs that show how a particular relation ties a group of concepts together, as in part-whole relationships.

The **Table edit windows** (5) display one of a number of tables describing a set of features that are part of the definition of the current edit object. The one displayed in (5) is the Local slot edit window, which has one line for each locally defined slot of the concept named. This is the normal table to see when editing a concept, although there are a number of others. Columns in the slots table show the source (where it was inherited from) of the slot, the name of the slot (which is also the name of the *role* or relation that the slot represents), the slot's value and number restrictions default value, and a textual description of the slot.

The **Features Commands window** (6) is a menu containing commands for displaying attributes of concepts or roles. There is a Concept Features Commands Window and a Role Features Commands Window. This menu is used to change what is displayed in a table window, or to display additional windows for a concept or role.

The **Editor Interaction Window** (7) is a Lisp Listener with a KREME command interpreter running. (Normal LISP expressions can also be typed into this window, by hitting the <SUSPEND> key first. KREME commands (like the ones displayed in command menus) can be typed in to this window directly, or they will appear when you click on a command from a menu. This window is also used when KREME needs to ask the user for information. Like all of the other windows, this window can be scrolled backward and forward through a history of the current session using the scroll bar at the left.

The **Mouse Documentation Window** (8) is always visible on Lisp machine screens. This is where you look to see what the mouse will do if you click one of its buttons.

2.3 Window Operations

There are a set of operations that enable you to move and reshape any window on the screen. These operations are accessed by clicking the mouse in the top margin (the solid area at the top) of the window you wish to operate on. Clicking the left button causes the border around the window to be highlighted. The window can then be repositioned by moving the mouse (which moves the highlighted border) to where you would like the window to be placed. Click the window again to put it there. (You can also do the same thing by clicking and holding the left button down, dragging the window to where you want it and releasing the button.) The middle mouse button acts similarly to reshape the window. Clicking that button on the margin area causes a carat (^) to appear. Move the mouse to the side or corner you would like to move, then hold the left button down while you stretch or shrink that side or corner. One very useful feature of the shape and move commands is that by pressing and holding the SHIFT key while moving a window stops movement temporarily, and allows the window to be reshaped by stretching or contracting the lower left corner. Lifting the SHIFT key continues moving it. Conversely, while the window is being reshaped, the SHIFT key allows the window to be moved. Releasing the SHIFT key continues reshaping.

The right button on the mouse, when over the margin of a window, contains a menu of commands that include the operations available on the left and middle buttons[2]. In this case, that menu also includes the following operations:

- **Bury** the window - that is, place it beneath all windows that it is currently on top of.

- **Expand** the window - reshape it to make it take up as much space as it can without overlapping other exposed windows.

- **Expand Horizontally** - reshape the window horizontally to take up as much space as it can without overlapping other exposed windows to the left and right of it.

- **Expand Vertically** - reshape the window vertically to take up as much space as it can without overlapping other exposed windows above and below it.

- **Hardcopy** - Send an image of the window to the printer.

- **Iconify** - Shrink the window to the size of the margin area. Once this is done, the operations available on this menu include only restoring it, killing, moving and burying the window.

- **Kill** the window - that is make it disappear completely. You will be prompted with a confirmation box if you use this option. To complete the Kill operation, click inside this second box.

- **Move** the window - This clicking and draging moves the window to a new place on the screen

- **Move Constrained** - Clicking this option again causes the carat to appear. Click the left button near either side to make the window move horizontally, or against the top or bottom to make it move vertically.

- **Search** - Clicking on this option and then typing a sequence of letters does an incremental search for that string within the window. The letters you type will be highlighted as they are found. This is a way to scroll the window to display some symbol that is off screen, or simply to find something that is visible.

---

[2]This is generally true wherever the mouse is. There is always a menu activated by clicking the right button of the mouse, and it always contains, among others, the operations found on the left and middle buttons.

- **Set character style** - This option is used to change the typeface used within the window, making it larger or smaller. A menu of choices appears showing the typefaces and how they would appear.

- **Shape** - This is the same as the <Middle Mouse> button action.

- **Shrink this window to expose others** - At times a window is slightly too big, and it causes other windows to be obscured, making those other windows inoperative. This option provides a means of shrinking the window just enough to fix this problem, without you having to manually move and reshape it.

- **Menu of Display Operations** - This command brings up a menu of operations on the window, as described below.

2.3.1 Window Display Operations

Each window can, in principle, be used to display a number of different things, by changing the command controling what is displayed in that window. The normal use of this feature of the system is to change a table window from displaying one aspect of an object to another, as to change from displaying **Local Slots** to display **All Slots**, including inherited ones. To make this possible, each window is associated with an object, called the **focus** of the window, and a **command** that determines what kind of information displayed about the focus object.

When a command like **Display Local Slots** is issued for a concept, KREME must choose a window to display those slots in. Normally, one window is designated as the **default output window**. This window is designated by an asterisk (*) appearing after the title of the window in its top margin. If there is no default output window, KREME assumes that you want to build a new window, and a rectangle will appear, that can be positioned by moving the mouse. Clicking left will cause the window to appear at the current position of that rectangle, filling it completely. This rectangle is the same as the one that appears when a **Move** window command is issued, and so you may also reshape the window that will appear by pressing SHIFT, before you place the window in its final position. This new window will become the default output window.

Windows can also connected to each other, if they are all displaying aspects of the same object. So, for example, in figure 2.2.1, all of the windows in the upper part of the screen are displaying aspects of the concept for a CAR. This group of windows is called a **window cluster**, because they are all *tied* to each other. Any command that changes the focus of one of these windows changes the focus object for all of them.

For example, if another **Edit Concept** command is issued for the concept ENGINE, the default will be to use as a display window the Concept Attributes window currently displaying the attributes of the concept CAR. This command will change the focus object of that window to be ENGINE, and simultaneously change the focus of all of the other windows in the cluster (that is, all windows tied to it, directly or indirectly) to also be ENGINE. The effect of this is to have all of the displays in the top half of the screen all show things about ENGINE. The graph window, which was associated with the command **Graph Concept**, will now display a graph of the abstractions and specializations of the concept ENGINE. The Local Slots window will display the local slots of ENGINE, etc.

This behavior of the **Edit Concept** and **Edit Role** commands can be changed by using the **:Display** keyword

command option on the command line in the interaction window. When there are no Edit Concept clusters of windows displayed, the default for the :Display keyword command option is **New Display**, e.g., to create a new one. Otherwise the default is to reuse one that is already there, as described in the last paragraph. If you do not wish the **Edit Concept** command to reuse the existing display windows, you can cause a new cluster of windows to be created by providing the :Display keyword command option with value **New Display**. Or, if you wish to have the effect of a command reuse windows that it would not normally use, you may issue the :Display option and point at one of the windows in the cluster you wish to use.

3. Extended Classification in KREME

One of the first tasks we addressed in Phase Two involved the extension of KREME's classifier to deal with more complex restrictions on slots. Our goal was to give knowledge engineers the enriched expressive power they needed to specify more precisely the constraints on slots in a concept's definition. We based our enhanced slot definition language on those found in commercial frame language systems like Intellicorp's KEE [Intellicorp 84]. Our goal was to demonstrate that such techniques could be used with frame language systems like KEE. However, we wanted to do this without giving up the important role played by the KREME classifier. The difficult part of this task was extending the classifier, so that its role in knowledge base maintenance was preserved. This meant enabling it to reason over restrictions composed of boolean combinations of concepts, as opposed to simple constraints that referred to single concepts. This chapter briefly reviews that work.

The success of the effort to extend the classifier depended in large part on the unique design of the KREME classifier, a design that was also crucial to the success of KREME in doing *re-classification* and *generalization*. KREME's classifier, unlike the classifier of its predecessor, NIKL [Moser 83], enables users to modify and re-classify concepts during an editing session.[3] This means that there is no need to edit textual definitions of the concepts or frames used in an application and then reload the whole knowledge-base in order for classification to occur correctly. The KREME classifier acts somewhat like a *truth maintenance system*, in that it takes revised definitions of concepts and propagates the effects of redefinition to all related definitions in the knowledge-base. For a full description of the KREME classifier, readers should refer to [Abrett and Burstein 87].

The modular design of the KREME classifier, and the introduction during Phase One of an independent hierarchy for slot restrictions made the job of extending the slot restriction language and classifier much easier. In order to allow redefinition to occur in a reasonable period of time, and in anticipation of the extension of KREME to a language with more complex slot restrictions, the classifier was redesigned mid-way through phase one of the project to include the explicit caching of the subsumption relations between the *slots* of different concepts, as well as the relations between the concepts themselves. In effect, KREME builds a classification hierarchy of slot restrictions that exists separately from the concept and role hierarchies Figure 3-1 shows how these hierarchies are related. Each slot on a concept is classified separately as to how its value and number restriction relates to others that have been used elsewhere in the concept network. Slots are also related by virtue of their slot name, which is the name of the *role* or relationship that they denote. Since roles also live in a subsumption hierarchy, as with the relation HAS-PROPERTY and its child COLOR-OF, then the slots that refer to those relations are also related.

In the KREME classifier, each component language definition object was made responsible for answering subsumption questions in order to relate two objects of that type. This meant that we could extend the slot restriction language and make classification work for that extended language simply by adding special rules for testing the subsumption relations between two complex slot restrictions, in order to determine their relative positions

---

[3]The classifier in LOOM [MacGregor 88], another system based on NIKL, and developed at ISI, now also does reclassification.

Figure 3-1: Subsumption Hierarchies in KREME Frame Language

within the subsumption hierarchy. The bulk of the classifier algorithm, for dealing with the global placement of new objects in a hierarchy, remained unchanged, as did the classification of concepts that used these extended restrictions.

Internally, complex restrictions were stored in a cannonical form, so that they could be efficiently compared. Thus, when a new value restriction was entered, it was converted to conjunctive normal form (CNF), with some rewriting to reduce the complexity of the expression, where possible. Finally, we specified a set of rules for comparing two boolean descriptions in CNF, and answering the key question, whether one subsumed the other or not. This can be a very expensive procedure in general, since the problem is NP-complete. However, for expressions involving less then 10 or so concepts, we found speed was not an issue, and we added some heuristic rules to this procedure that would look for shortcuts to answering this question, where a standard algorithm would have been unnecessarily slow.

Unfortunately, it is theoretically impossible to make the resulting algorithm logically complete. It cannot *always* resolve correctly that one concept subsumes another. It is, however, never going to incorrectly state that a subsumption relation exists when that is wrong, or it goes in the other direction. It simply gives up on particularly difficult cases and states that the two restrictions are incomparable. This means that the classifier will essentially create siblings for such cases. This also means that the classifier for concepts may potentially miss a subsumption relation that it could have inferred, in theory. However, for the vast majority of cases, the system functions well, and provides the knowledge base builder with more than he entered explicitly.

Below is an example of what the classifier does with these extended restrictions. We defined the concepts CAR, RED-CAR, BLUE-CAR, RED-AND-BLUE-CAR, and RED-OR-BLUE-CAR. Each of the colored cars was defined as a *non-primitive* specialization of CAR, with a different value restriction on the COLOR-OF slot. Since they were all non-primitive, the classifier is free to assume that the concepts are completely specified by their definitions, and it tries to relate them based on their defined properties. As their names indicate, the RED-CAR had a value restriction (VR) of RED on the slot COLOR-OF, the BLUE-CAR had VR BLUE on that slot, the RED-AND-BLUE-CAR had VR (AND RED BLUE), and the RED-OR-BLUE-CAR had VR (OR RED BLUE) on the same COLOR-OF slot.

In general, there could have been many concepts that used these same restrictions on COLOR-OF. However, the relationships *between* these restrictions are computed only once. The classifier takes these restrictions and relates them to each other. Each restriction is placed in the restriction lattice so that it is beneath restrictions that subsume it, and all equivalent restrictions are classified the same. That is, concepts with slots using equivalent restrictions have those restrictions unified in the restriction lattice. Even though there could have been many concepts that used these same restrictions on COLOR-OF, the relationships between different restrictions are thus computed only once, and each is then placed in the restriction hierarchy under the more general restriction COLOR-OF:(A COLOR), which was defined for CAR. Figure 3-2 shows a KREME graph of this portion of the restriction hierarchy.

Figure 3-2: A KREME Restriction Hierarchy Graph with Boolean Restrictions

As you can see from this graph, the restriction (OR BLUE RED) is more general than the restrictions RED

and BLUE separately, since the class of things that is *either* RED or BLUE includes all red things and all blue things. The restriction (AND RED BLUE), since it is only true of things with both colors, is more specific than *both* the class of things with color RED, and the class of things with color BLUE.

Once the KREME classifier had established these slot restriction relationships, it turned to the classification of the concepts that used those slots. All of the colored cars were defined as non-primitive concepts differing only by the restriction on the COLOR-OF slot. They were also defined as specializations of the concept CAR. In a system without a classifier, they would all be placed as direct descendents of CAR. However, the classifier, knowing how they all differed only in one slot restriction, could detect that some were specializations of the others, and establishes a hierarchy for those concepts that parallels the hierarchy of their restrictions on COLOR-OF. This is shown in figure 3-3 below.

Figure 3-3: Resulting KREME CONCEPT Hierarchy Graph

4. KREME and SFL in CLOS

In preparation for use of KREME on other platforms, notably SUN workstations, we undertook the conversion of KREME from its original implementation in terms of Symbolics FLAVORS objects to the PCL object system, the preliminary implementation of CLOS developed at Xerox PARC. CLOS stands for the Common Lisp Object System [Bobrow et al. 88]. The conversion of the internal frame language definition objects and classifier was a relatively straightforward translation of one object system to another. The new system, with some optimization, appears to work about as fast on a SUN 4 as the original FLAVORS system did on a Symbolics Workstation. This makes it well suited to use in that environment. A preliminary conversion of the revised interface and editing environment has also been completed. The SUN interface is built using CLIM, the Common Lisp Interface Manager developed by International Lisp Associates in conjunction with a number of LISP vendors and developers.

For another project at BBN, Glenn Abrett, the principle implementer of the original version of KREME, and others, developed a "stripped down" version of the original KREME language and editor that focused on rapid editing and instantiation in a production environment, without the overhead of full classification. This system is called SFL for Simplified Frame Language. SFL, together with a revised version of the KREME procedures language and editor called TARL, the Tactical Action Representation Language, are the knowledge acquisition and representation tools for a complex multi-agent simulation facility that is used in the Semi-Automated Forces system for SIMNET, a battlefield simulation and training environment. These representation tools are discussed in the next chapter, to show one of the uses that the KREME technology has been put to.
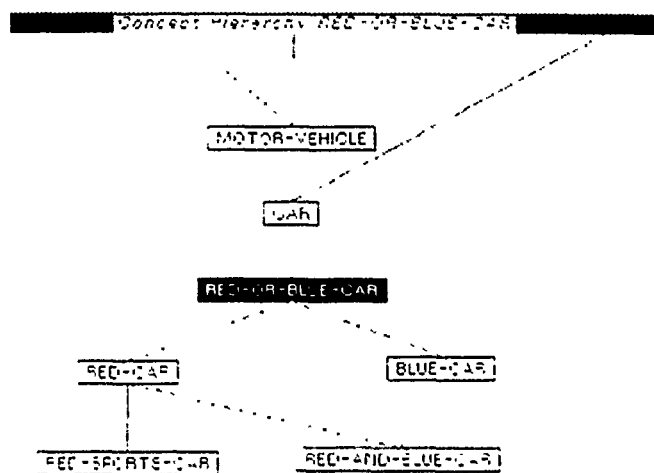
Our work converting KREME and SFL to run under CLOS raised an interesting possibility. Because CLOS has an accessible *meta-object protocol*, it was considered possible to write a version of the KREME language that was written directly as a set of meta-classes in the CLOS environment, providing a truly integrated object system with SFL/KREME Frame semantics for slots. This experiment was undertaken by Tom Reinhardt, and is reported in full in [Reinhardt and Burstein 89], a paper presented at the 1989 OOPSLA conference on object-oriented programming. This section gives a brief overview of points made in that paper. For further discussion see also the Phase Two Technical Report [Burstein et al. 89a].

4.1 A Meta-class based Frame system

Many efforts at BBN have considered the use of CLOS when building a knowledge base for a production-oriented knowledge based system. This investigation looked at one way to ease their entry into that environment. We endeavored to introduce the frame language SFL directly into the CLOS environment using the meta-object protocol. Unlike Kreme, SFL does not attempt to maintain a completely classified hierarchy of concepts and relations. Instead SFL accepts the (parent-child or subsumption) hierarchy as defined by the user, and ensures that the properties, i.e., the slots, are inherited correctly using the completion algorithm from KREME. Whereas

completion was a subtask of the original Kreme language classifier, it is the primary task of SFL. We were able to develop a version of this algorithm to replace the effective slot computation mechanism of the standard CLOS meta-class, thereby producing a true CONCEPT meta-class.

4.1.1 Integrating Objects and Frames

From numerous projects currently underway at BBN we have seen a requirement for a knowledge representation and/or acquisition systems that are similar in (1) their use of large amounts of data requiring a fast, easily verifiable knowledge representation language, and (2) their need for fast *instantiation* of objects from their underlying descriptions. Just as importantly, though, they differ in the size and complexity of their underlying knowledge bases, and in their need to *incrementally evolve*. This, in turn, implies their need to operate over incomplete, possibly inconsistent data.

Of these items, the last, the need or desire to *incrementally* define and edit these large knowledge bases in the absence of total knowledge, is critical in evaluating the desirability and feasiblity of the integrating frames into CLOS.

The Meta Object protocol, as described in Chapter 3 of the **Common Lisp Object System Specification** [Bobrow et al. 88], provides a general mechanism for extending the CLOS interpreter.[4] A new language, SFML (the Simple Frames Meta Language), has been implemented within this mechanism. We have shown that by defining specific protocols at this level, the desired behavior, i.e., the salient properties of SFL, can be bootstrapped in a general and attractive manner. Moreover, SFML provides functionality above and beyond that provided by the original SFL, specifically,

1. It permits incremental definition of knowledge bases;

2. Designers may quickly and predictably particularize its behavior through the redefinition of generic methods;

3. It provides a builtin instantiation mechanism that has and will continue to be refined over time.

4. It is envisioned that upon acceptance of the Meta Object Protocol by the Common Lisp Language Committee, X3J13, SFML will be completely portable and therefore available to a variety of traditional as well as emerging architectures.

These properties are, by in large, a direct result of having implemented SFML within the CLOS metaobject protocol.

---

[4]It should be noted that although the Common Lisp Object System Specification has been accepted by the Common Lisp Language committee, X3J13, Chapter 3, outlining the Meta Object protocol, has not as of this writing.

4.2 Design and Implementation

In order to follow in detail the design and implementation process, a detailed understanding of the meta-object protocol is necessary. Interested readers should refer to [Reinhardt and Burstein 89] and [Bobrow et al. 88] for details. Here, we just give the basic ideas behind the process.

The CLOS Meta-object protocol is a set of modifiable behaviors associated with the definitions of all *objects*. Normally, the classes of objects defined for an application are derived from STANDARD-CLASS, in much the same way FLAVORS provides a builtin definition mechanism for building classes of flavor objects. The difference is that one can build modified versions of STANDARD-CLASS quite naturally, where the definitions of FLAVORS cannot be similarly modified. Thus, the object classes for CONCEPT, ROLE, SLOT and RESTRICTION in the original KREME system were defined as *flavors* (See Figure 2-2.), and in the straightforward conversion to CLOS, they became *classes*, based on the meta-object STANDARD-CLASS.

An intermediate step between SFL-CLOS and SFML was the introduction of instantiation of concepts in SFL-CLOS by specializing STANDARD-CLASS to a new meta-object that simply combined (mixed together) STANDARD-CLASS and CONCEPT, to form STANDARD-CONCEPT, and declared that all slots stored in instances of STANDARD-CONCEPT should become instantiated by the mechanisms of the original STANDARD-CLASS. This effectively made all concepts be classes in the CLOS sense, for the purpose of instantiation. The approach, while successful, required a lot of unnecessary overhead. Essentially, two object/frame mechanisms were invoked for each concept defined. It was clearly a very inefficient approach to instantiation.

In SFML, a different approach was taken. Modified versions of STANDARD-CLASS were developed that combined the repeated functionality much more carefully. The definitions of slots in the KREME/SFL sense were carefully integrated into the CLOS slot mechanisms, by specializing the CLOS STANDARD-SLOT meta-class. In fact, several varietys of slots were developed, one which included only the additionalinformation about number restrictions, value restrictions and defaults found in the KREME/SFL language, another which also had capabilities of *demons*.

The CONCEPT meta-class was then defined to use these new kinds of slots. This meant replacing the methods on the meta-object STANDARD-CLASS that computed the effective slots for a class with new versions that used the KREME/SFL completion algorithm to do the same job. This was only possible because of the design of the meta-object protocol. What was required was the carefully replacment of one slot-computation mechanism for another by defining a specialized version of the method COMPUTE-EFFECTIVE-SLOTD for the new, specialized version of STANDARD-CLASS. There are a number of other details involved in this kind of an experiment with the meta-object protocol, but they go beyond the scope of this report.

4.3 Results of the Experiment

This is clearly only an experiment in the unification of frames and objects. However, we feel it is an important one, since CLOS is rapidly becoming the standard for object systems in Common Lisp. Unfortunately, as of this writing, the meta-object standard has not been accepted as a standard, so we cannot yet rely on the portability of that mechanism to all platforms and Common Lisp systems.

While we were successful in getting the merger to take place, there are a number of efficiency issues that remain. The original FLAVORs version of SFL took great pains *not* to develop all of the structure necessary to instantiate each and every concept, so as to avoid the overhead that that introduced. In a typical system, only the most specific concepts are ever instantiated. The more general ones are there only to support the consistency and inferential capabilities required of a knowledge base. On the other hand, the merger of the SFL and CLOS produced a system in all concepts/classes are instantiable objects, with a reintroduction of all of the overhead that that implies. Were we to make this into a true production system, this issue would have to be addressed more carefully.

On the other hand, the whole CLOS system continues to become more efficient, so that in the future, this overhead may not be a burden. We forsee a day when a system like SFML will become the standard for production object/frame systems.

5. Applications of KREME Technology

During the period of time that Phase Two of the Knowledge Acquisition project was active, KREME editor tools and derivatives of KREME have been used for a number of applications at BBN and elsewhere. This chapter looks at some of those uses. The first section gives a chronology of the projects that have used KREME tools. The second section looks at SFL and TARL, spinoffs of the KREME frame and procedures editors respectively that were an important component of the Semi-Automated Forces project within the DARPA-funded SIMNET battlefield simulation and training system. We conclude the chapter with a description of a project investigating the use of KREME in an ongoing knowledge acquisition and refinement activity, the development of indices for an on-line document library.

5.1 KREME Applications and Spinoffs

We begin with a bit of a chronology. There have been a number of applications that, directly or indirectly, have used KREME environment tools and representation techniques over the last three years. We list the major ones here, in order of time of their development.

- January - April 1987 -- A version of the original KREME editing environment was grafted onto NIKL for use in the JANUS natural language system, based on an earlier grafting of the environment onto KEE. This effort was only partially successful because of the inability of NIKL to *reclassify* definitions.

- August 1987 - December 1988 -- SFL, the Simplified Frame Language, and TARL, the Tactical Action Representation Language, and editors for both languages were developed, based directly on the KREME frame and procedures languages. These tools became part of the event-driven, object-oriented simulation system called SPROKET.

- June - September 1988 -- The new version of KREME in CLOS with the new interface design was implemented.

- July 1988 -- A browser and class editor for CLOS was developed directly from the new KREME interface.

- August 1988 -- A CLOS browser based on the new KREME and a revised KREME rule editor are used in an expert system for the IRS.

- September 1988 -- Work begins on the use of KREME in an document indexing and retrieval system to be called INDEXER.

- October 1988 -- The new KREME frame system and editing environment was adopted for use as part of a new Knowledge Acquisition facility for the JANUS natural language system.

- December 1988 -- A version of SFL in CLOS was developed, with the new KREME interface.

5.2 SFL and TARL

SFL and TARL are representation language editing environments directly based on KREME.[5] SFL, as has been described earlier, is essentially a version of the original KREME Frame Editor without the classifier with the addition of a Flavors-based instantiation mechanism that allows system developers to specify which concepts need to get instantiated, and which concepts should be defined with "external" flavors mixed into them. This second feature provides an extremely flexible mechanism for mixing programmed behaviors into concepts, so that they can be used as active objects in a knowledge based system. TARL is a languaage for representing both goals and procedures. It uses SFL to represent the *classes* that different goals and procedures fall into, and their attributes. Both TARL and SFL are a part of the SPROKET simulation environment.

The SPROKET simulation environment is based more on AI planning languages than other simulation languages in that the goals, plans to achieve goals, and actions undertaken as parts of plans are all represented at the knowledge level [Newell 81]. That is, they are described in a frame-based declarative language that can be reasoned over. The simulator is optimized to interpret these declarative, object-oriented descriptions of goals and plans.

The knowledge representation languages used by SPROKET are accessed through a powerful set of knowledge acquisition tools for browsing and editing declarative representations based on KREME. These editing tools are essential in designing complex plans and behaviors for simulated agents with a high degree of flexibility and accuracy.

SPROKET's procedure representation language makes the construction of procedures that can be simulated in parallel quite easy. Each of the agents is simulated as acting in parallel, and can pursue multiple independent goals in parallel. Agents can even perform several actions "simultaneously" if those actions are not mutually exclusive.

Figure 5-1 is intended to give a sense of how all of the pieces of the SPROKET environment fit together. There is an underlying representation language for frames, and other languages for constructing goals, their subgoal decompositions or plans, and procedures or scripts that can be used to achieve goals by performing contingency-bounded sequences of actions. These languages are closely tied to a set of editing environments that were built using tools developed for the KREME knowledge acquisition environment. Finally, there is the event driven simulator that takes the user-developed representations of agents, their goals, procedures and actions, and causes the agents to "walk through" simulated situations and try to achieve their goals.

Figure 5-1: SPROKET system overview

5.2.1 SFL

SFL does not use the KREME classifier, but it does use most of the rest of KREME, including one portion of the classifier, the slot completion algorithm that determines a concepts set of effective slots given its set of defined slots and defined parents. The primary function of completion in SFL is to determine the simplest logical expression for each slot's value and number restrictions, based on the slot's local definition and the restrictions that the concept inherits for that slot from its parents. Logically, these restrictions are the conjunction of their locally defined values and the restrictions inherited from slots implementing the same relation on parent concepts.

In principle and in fact for the first version of SFL, the relationship between SFL concepts and Flavors was established by shadowing the concept taxonomy with a duplicate flavor hierarchy. For every concept there was a corresponding flavor, whose mixins (component objects) corresponded to the concept's direct parents, and whose instance variables were each associated with one of the corresponding concept's slots (the slot names were identical, as were the default values)[6] Whenever a concept was defined a shadow flavor was defined along with it. An instance of a concept was in fact an instance of the corresponding flavor.

---

[6] There was one addditional slot on the flavor, containing a pointer back to the concept that it was a type of.

In the current version of SFL, this scheme has been modified for efficiency reasons to make the shadow flavor hierarchy much sparser than the concept hierarchy. Only those concepts that are actually going to be instantiated or have specific behaviors defined for them are actually shadowed by flavors. This eliminated a large amount of needless flavor definition and redefinition as a concept latice was under development, a tremendous savings in both time and space. Figure 5-2 illustrates this scheme.

This relationship between instantiable concepts and corresponding flavors gives SFL an efficient means of associating behaviors (methods) with its concepts' instances. However, in order to more clearly distinguish the *behavior* of objects in the *simulation* from their representations as concepts, concepts and flavors were by allowing concepts to be directly associated with "external" flavors, flavors not part of a concept's "knowledge level" definition, but mixed into the flavor used to instantiate the concept, providing it with additional runtime state information and behaviors.

The SFL Editor Interface is directly based on the original KREME interface. It is shown in figure 5-3.

5.2.2 TARL

The TARL language is used for representing the goals, plans and procedures that drive the simulated agents. These representations are at heart of the SPROKET environment, and the tools used to develop representations in these languages are a large part of what makes SPROKET so powerful. Figure 5-4 is an example of the editor interface to this set of representation tools, showing its strong connection to KREME.

TARL most closely resembles PRS, Georgeff and Lansky's procedural reasoning system [Georgeff and Lansky 86]. The portion of TARL that is used for representing plans to achieve goals as an AND/OR graph of subgoals is based directly on PRS. The major difference is that TARL provides a second layer for representing conditionalized sequences of low level actions that is not part of PRS.

Figure 5-5 shows how the different "layers" of TARL are related to each other, and to the actions that the simulator schedules and performs. At the top are the goals that the agent has and is trying to achieve. *Goals* represent things that need to be accomplished. As in PRS, a goal is characterized by a set of *achievement conditions* that describe the state of the world under which the goal can be said to have been achieved. Each goal is also associated with an *achievement plan* (hereafter, simply called a plan) that describes the alternative sets of subgoals and procedures that can lead to satisfying the goal's achievement conditions. Achievement plans (the second layer in from the edge in figure 5-5 are specified as an AND/OR tree of subgoals, where subgoals may simply invoke procedures. The default assumption in plans is that both conjunctive and disjunctive subgoals may be attempted "in parallel", although that assumption can be overridden by explicit "serialization" links in the plan graph.

Eventually, one must actually do something to achieve one's goals. Thus, plans generally bottom out in procedures that are like *scripts* [Schank and Abelson 77]. Procedures are the lowest knowledge level (declarative) description of how something in the simulated world is to be done. Procedures describe sequences of atomic

Figure 5-2: Relationship between Concepts and Flavors in SFL

Figure 5-3: The SFL Editor Interface

Figure 5-4: The TARL Procedure Editor

Figure 5-5:   Goals and Procedures in TARL

actions to be taken, tied together by temporal sequencing links into things that look like flow-charts. Actions are encapsulated LISP functions, that are performed by the simulator, and which modify the simulated environment.

5.3 A KREME Knowledge Base for Document Retrieval

BBN has been working for approximately a year developing a document indexing and retrieval system for agencies within the Department of Defense that are responsible for Technology Assessment. The overall goal of this project is to develop and provide prototypes of distributed information management tools to those agencies within DoD responsible for technology cooperation and exchange. These agencies have expressed a need in recent years to m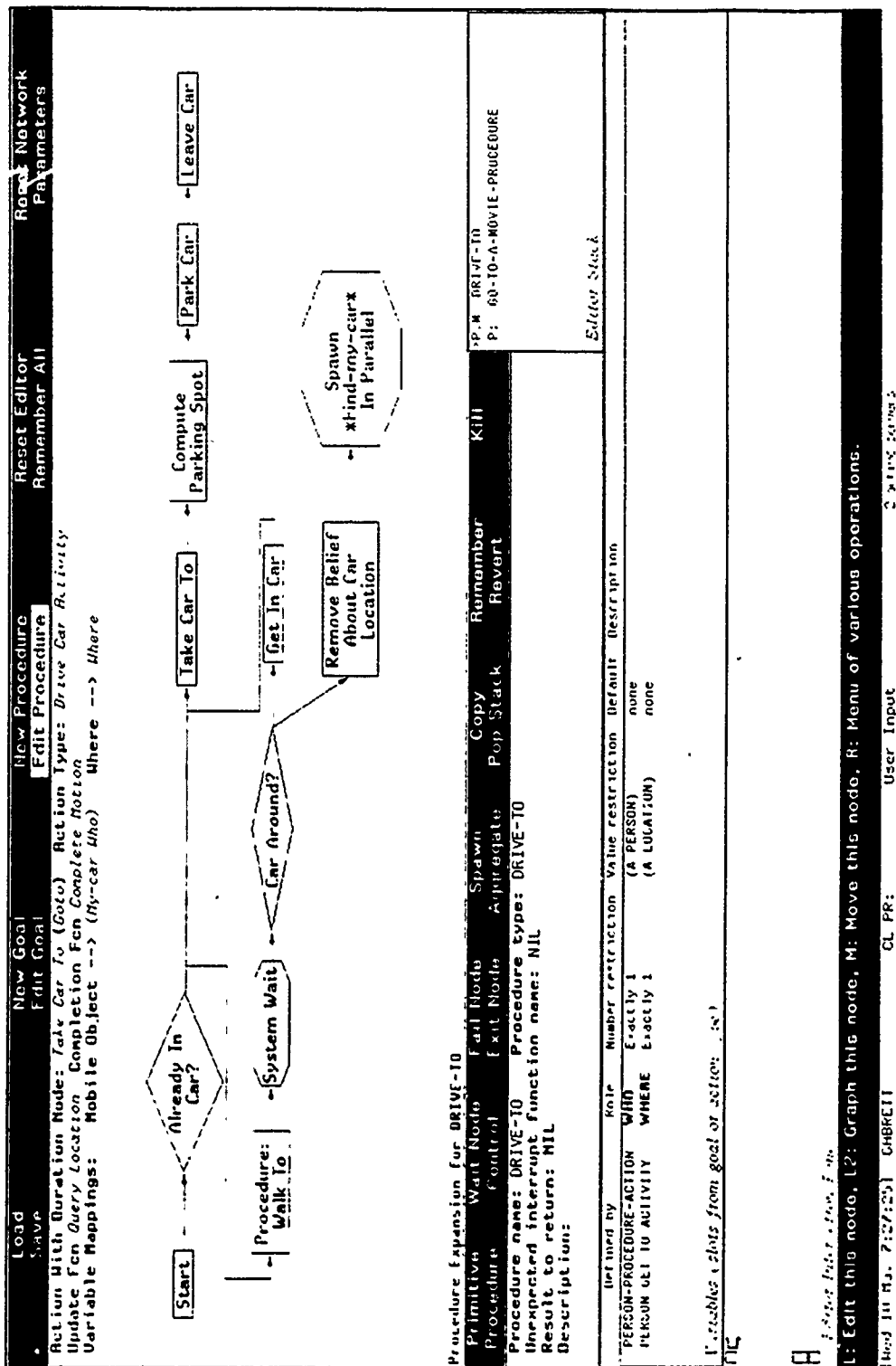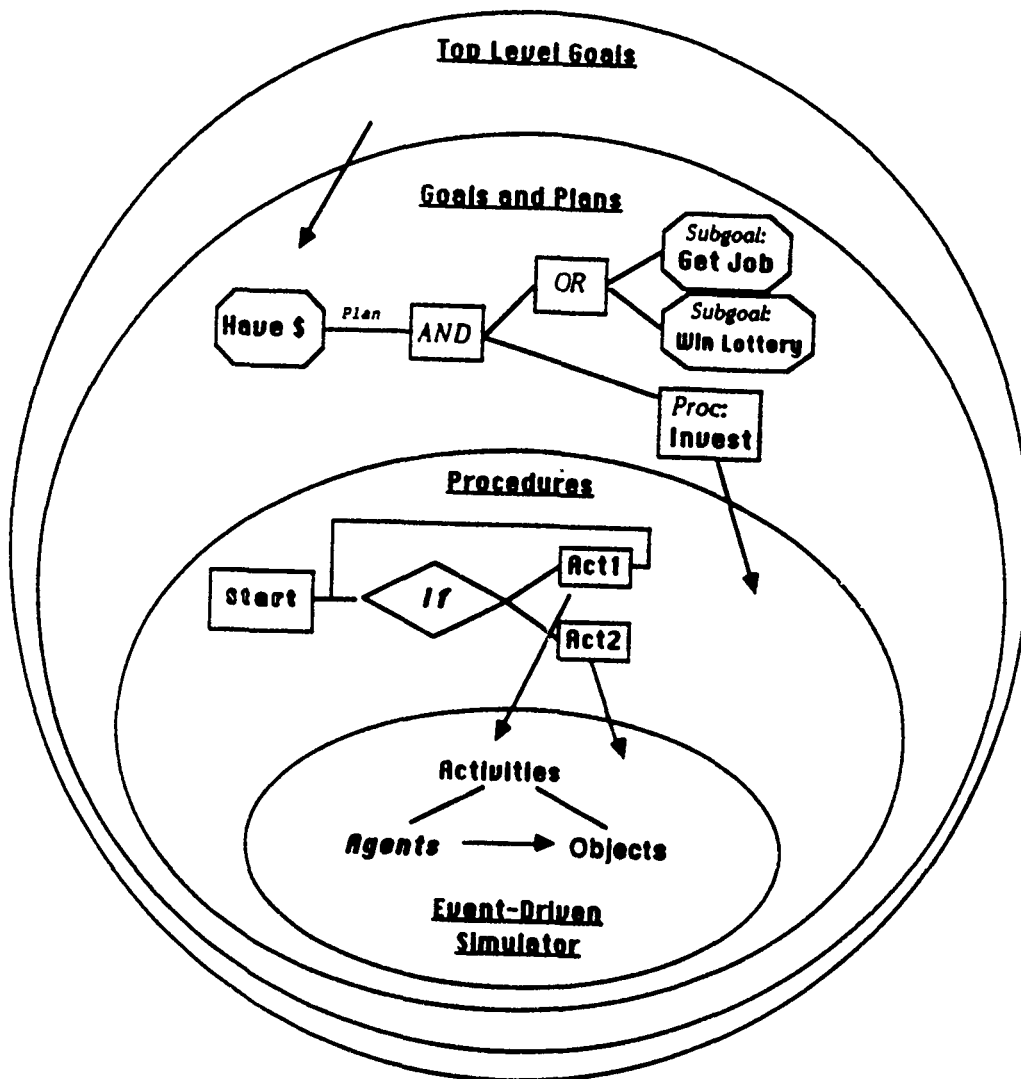ove from a primarily reactive mode into a more active mode in identifying technologies to be pursued, in establishing agendas for meetings, and in developing the memoranda of understanding which form the instruments of exchange. Additional tools will be added to the workstation environment we have already developed to support this active mode. One of these is a more knowledge intensive approach to document indexing, taking advantage of KREME technology.

In a short time the document databases and information resources of technology exchange operations within DoD will be vast. State-of-the-art automated document handling techniques, whose storage and retrieval algorithms are ignorant of the content of the documents, will be inadequate. Such techniques may include extremely clever exploitation of key phrases, statistical properties, indexing schemes, etc.; but they tend to confuse documents which use similar words in different in senses (or use different words, but are similar in content). The result is that they retrieve too many documents (or miss relevant documents). The technology exchange context requires a system which intelligently exploits knowledge of subject matter, people, projects, countries, agreements, etc; recognizes such concepts in texts; and assists human document handling by exploiting such knowledge.

5.3.1 DEARS

The DEARS (Document Entry And Retrieval System) was first developed to support the basic task of building a large-scale, on-line, document-library retrieval system for technology cooperation operations.

The development of the current DEARS system has involved the application of three key technologies: the document manipulation techniques embodied in the BBN Slate™ system, the knowledge acquisition techniques developed during the construction of the KREME system, and the distributed computing environment of the BBN Cronus system. This development also includes a well-developed infrastructure of fundamental information management components, intra-DEARS interfaces, and user interface components. Each of these components is briefly described below:

Document Identification and Location System. This system supports storage of the various data required to identify and characterize documents. Such identification information includes such information as author, document

type (e.g., briefing, article, correspondence, agenda, proceedings), DoD document serial number, physical location, date, relevant organization, etc.

Library Catalog System. This system supports cataloging of documents in a basic library card catalog fashion. The fundamental subject/title/author catalogs are supported, with extensions to allow users to easily create new catalogs according to domain or retrieval requirements (e.g., briefing catalogs, weapons catalogs, etc).

Relational Database System. This system is a general-purpose, commercial RDBMS with a Cronus-based distributed interface. It accommodates both C language and LISP language-based components, and is used as a general storage and retrieval resource for other system components.

Document Formatting System. This system preserves the original formatting of documents entering the system. It is a LISP component that analyzes the presentation format of a document. This analysis is recorded with the text of the document in the form of BBN Slate™ document format control statements. When a document is displayed in by BBN Slate™, it appears in its original layout.

Keyword System. This system provides storage and retrieval mechanisms for document/keywords tuples. Upon entry to the system, documents are scanned and the presence of keywords in the text is recorded. This component provides an additional tool for general document retrieval.

Graphical Interface Toolkit. This toolkit consists of a set of very high level routines for developing graphical user interfaces.

The completed DEARS system will combine three approaches to indexing and retrieval, each of which represents a phase in the development of the system:

1. standard techniques for users to manually index documents in "card catalogues", as represented by DEARS,

2. fully automatic techniques for indexing documents by word usage indices, also in the current DEARS system, and

3. semi-automatic AI-based techniques for conceptual indexing of documents using case-frame representations.

5.3.2 INDEXER

We are now developing a plan to go beyond the traditional kinds of electronic document indexing exemplified by the topic cataloging and inverted word indexing techniques available in our first prototype, the DEARS system. Our primary goal in this research program is based on what we feel is a pressing need for exploration of the use of AI knowledge representation techniques in the pursuit of more *machine intelligent* document indexing and retrieval behavior. The strategy we will employ is based on the supposition that by encoding summaries of document content in declarative representations built on a lattice of *concepts and relations*, one may provide far more accurate retrieval behavior and account for more of the variations in expression that occur in language than is presently

possible using techniques that rely on the appearance of particular words or even sets of words. Our approach is not to throw away what can be achieved by those techniques, but to *augment* them with these more precise forms of indexing.

The results of our effort will be a new system, built as a companion to the DEARS system, that will provide a much more sophisticated set of mechanisms and interface tools to support human indexers of documents, and an "expert system" of general heuristic and domain dependent rules to guide searches of the document concept space. This system we call INDEXER, which stands for "Intelligent Document Encoding and indeXing for Enhanced Retrieval".

In implementing this strategy, we face the problem of encoding the content of documents to a "sufficient" degree without the availability of a fully general natural language understanding technology. Our approach is to provide a version of sophisticated user-interface that will automatically recognize most of the known terms and key phrases of the domain, and allow human indexers to deal more quickly and thoroughly with the ambiguities and questions of prioritization or relevance of different portions of the text.

This interface will be closely coupled to a version of KREME running on a SUN workstation. All of the concepts and relations that are used for indexing will be represented in KREME, and these representations will then be used to index documents, and clusters of documents.

5.3.3 Initial Study

In preparation for the full scale development of this, we have worked with the revised KREME system, developing an initial knowledge base of terms that will support indexing of documents in the technology assessment domain. A graph of this concept base is included in Appendix B. We are now working to connect a key phrase recognizer to this KREME knowledge base.

6. Conclusion

This report has reviewed work done under Phase Two of the BBN Knowledge Acquisition Contract. The overall goal of this project was to build a versatile experimental computer environmen for developing and maintaining large knowledge bases. We have pursued the goal along several complementary paths. First, we have constructed (several versions of) a flexible, extensible Knowledge Representation Editing and Modeling Environment (KREME), in which different kinds of representations could be developed and maintained. In building and equipping this "sandbox" we have been adapting and experimenting with techniques which we think will make editing, browsing and consistency checking for each style of representation easier and more efficient, so that knowledge engineers and subject matter experts can work together to build significantly larger and more detailed knowledge bases than are presently practical.

The second path has been in investigating a variety applications of these techniques, bringing different representational and user interface constraints to bear. Out of our preliminary efforts, a number of applications have grown that use many of the component pieces of software found in the original KREME system. Chapter 5 has looked at some of these.

A primary focus during Phase Two of the project has been improving the flexibility, from a user's point of view, of the KREME system, and also the portability and technology transfer opportunities for KREME technology. This is evidenced by the new interface described in chapter 2, our extension of the language semantics in chapter 3, and in our conversion of KREME to CLOS and investigation of a full integration of KREME semantics with the CLOS meta-object protocol, as discussed in chapter 4.

Given the range of experiences we have had with KREME in the past two years, we expect that much of the technology developed on this project will find its way into new and exciting applications for DoD in the years to come.

Appendix A
Test Plan

Loading KREME from Cassette Tape

Each site can test KREME by loading KREME from tapes according to the directions in this Appendix and then editing the sample networks provided on the tape. Once KREME has been loaded, the document The KREME User's Manual BBN Report 7175 provides instructions on how to edit and create knowledge bases using KREME.

KREME requires a Symbolics machine with Genera 7.2 or better already installed. If your machine has no tape drive, you will have to read the tapes on another machine that does have one and then transfer the files to the machine you wish to use.

There are two tapes provided. One is a Carry tape with some predefined *translations* files. This tape should be loaded first using the (tape:carry-load) command as described in Volume 0 of the Symbolics manual. This tape contains sample networks which should be placed in a directory with logical name NEWKREME:DEFS;*.*, and real name YOUR-HOST:>newkreme>defs>, where YOUR-HOST is the machine that will house all of KREME system. The other files are files to go into the SYS:SITE; directory. These are translations files for the component systems of KREME. Each of these SYS:SITE;*.TRANSLATIONS files must be edited to replace the machine name CONGER: with the name of your local machine that is to hold the KREME system files.

The second tape contains all of the files comprising the KREME system this tape should be placed in the tape reader after the edits described above have been made. Then, by issuing the **Restore Distribution** command (also found in Volume 0 of the Symbolics Manual) the tape will be loaded into the host machine that you specified in by replacing the CONGER: in the SYS:SITE;*.TRANSLATION files.

Once all files have been loaded, simply do the following two commands:

```
Load System CLOS :version Newest
Load System NewKREME :version Newest
```

When this operation has completed, simply hit <Select>-K, and you will be in the KREME system. Once this is done, follow the directions in the KREME User's Manual.

DISTRIBUTION LIST

RADC/COES   10
ATTN: Sharon M. Walter
Griffiss AFB NY 13441-5700

BBN Systems and Technology Corp   5
10 Moulton Street
Cambridge MA 02138

RADC/DOVL   1
Technical Library
Griffiss AFB NY 13441-5700

Administrator   2
Defense Technical Info Center
DTIC-FDAC
Cameron Station Building 5
Alexandria VA 22304-6145

Defense Advanced Research Projects   2
    Agency
1400 Wilson Blvd
Arlington VA 22209-2308

HQ USAF/SCTT   1
Washington DC 20330-5190

SAF/AQSC   1
Pentagon Rm 4D 269
Wash DC 20330

Naval Warfare Assessment Center   1
GIDEP Operations Center/Code 30G
ATTN: E Richards
Corona CA 91720

HQ AFSC/XTH
Andrews AFB MD 20334-5000   1

HQ SAC/SCPT
OFFUTT AFB NE 68046   2

DTESA/RQE
ATTN: Mr. Larry G.McManus   1
Kirtland AFB NM 87117-5000

HQ TAC/DRIY
ATTN: Mr. Westerman   1
Langley AFB VA 23665-5575

HQ TAC/DOA
Langley AFB VA 23665-5554   1

WRDC/AAAI-4
Wright-Patterson AFB OH 45433-6543   1

WRDC/AAAI-2
ATTN: Mr Franklin Hutson   1
WPAFB OH 45433-6543

AFIT/LDEE
Building 642, Area B   1
Wright-Patterson AFB OH 45433-6583

WRDC/MTEL
Wright-Patterson AFB OH 45433   1

AAMRL/HE
Wright-Patterson AFB OH 45433-6573   1

AUL/LSE
Bldg 1405   1
Maxwell AFB AL 36112-5564

HQ ATC/TTOI
ATTN: Lt Col Killian   1
Randolph AFB TX 78150-5001

AFLMC/LGY
ATTN: Maj. Shaffer   1
Building 205
Gunter AFS AL 36114-6693

US Army Strategic Def   .
CSSD-IM-PA   1
PO Box 1500
Huntsville AL 35807-3801

Ofc of the Chief of Naval Operation
ATTN: William J.Cook   1
Navy Electromagnetic Spectrum Mgt
Room 5A678, Pentagon (OP-941)
Wash DC 20350

Commanding Officer
Naval Avionics Center   1
Library D/765
Indianapolis IN 46219-2189

Commanding Officer
Naval Ocean Systems Center   1
Technical Library
Code 9642B
San Diego CA 92152-5000

Cndr
Naval Weapons Center   1
Technical Library/C3431
China Lake CA 93555-6001

Superintendent   1
Code 1424
Naval Postgraduate School
Monterey CA 93943-5000

Space & Naval Warfare Systems Comm   1
Washington DC 20363-5100

CDR, U.S. Army Missile Command   2
Redstone Scientific Info Center
AMSMI-RD-CS-R/ILL Documents
Redstone Arsenal AL 35898-5241

Advisory Group on Electron Devices   2
201 Varick Street, Rm 1140 .
New York NY 10014

Los Alamos National Laboratory   1
Report Library
MS 5000
Los Alamos NM 87544

AEDC Library   1
Tech Files/MS-100
Arnold AFB TN 37389

Commander, USAG   1
ASQH-PCA-CRL/Tech Lib
Bldg 61801
Ft Huachuca AZ 85613-6000

1339 EIG/EIT   1
Keesler AFB MS 39534-6348

AFEWC/ESRI   3
San Antonio TX 78243-5000

485 EIG/EIR
ATTN: S Buzinski   1
Griffiss AFB NY 13441-6348

ESD/XRR
Hanscom AFB MA 01731-5000   1

ESD/AVSE
ATTN: Capt Lesieur   1
Hanscom AFB MA 01731-5000

ESD/SZM
Hanscom AFB MA 01731-5000   1

SEI JPO
ATTN: Major Charles J. Ryan   1
Carnegie Mellon University
Pittsburgh PA 15213-3890

Director NSA/CSS
T513/TDL   1
ATTN: D W Marjarum
Fort Meade MD 20755-6000

Director NSA/CSS
W157   1
9800 Savage Road
Fort Meade MD 21055-6000

NSA
ATTN: D. Alley   1
Div X911
9800 Savage Road
Ft Meade MD 20755-6000

Director
NSA/CSS   1
W11 DEFSMAC
ATTN: Mr. Mark E. Clesh
Fort George G. Meade MD 20755-6000

DoD   1
R31
9800 Savage Road
Ft. Meade MD 20755-6000

DIRNSA   1
R509
9800 Savage Road
Ft Meade MD 20775

Director   1
NSA/CSS
R03
Fort George G. Meade MD 20755-6000